Uniform Type Identifiers and Launch Services

Session 107

These are confidential sessions—please refrain from filming or taking pictures.

Application Technologies

Uniform Type Identifiers and Launch Services

Christopher Linn
Senior Software Engineer

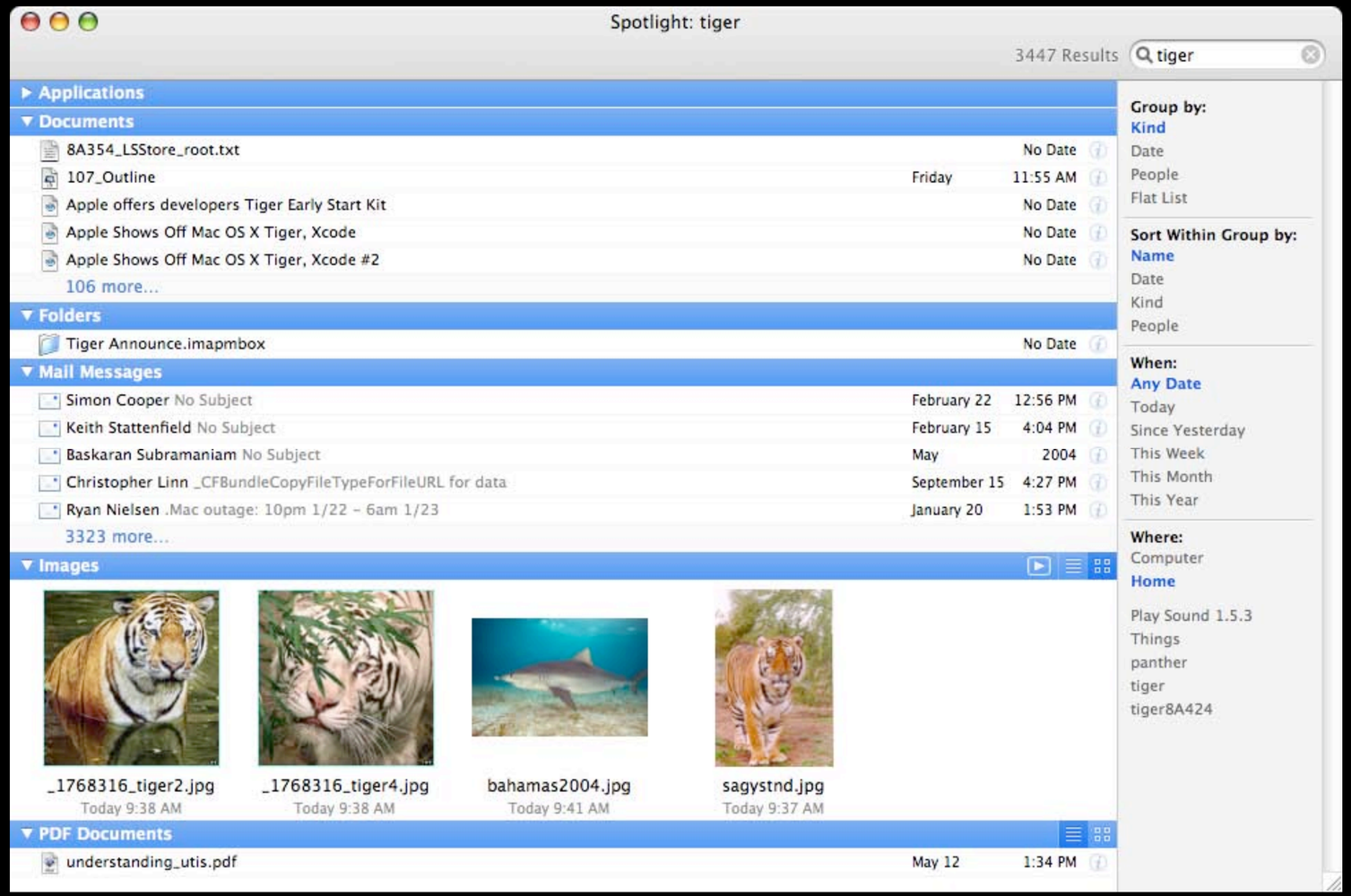
http://developer.apple.com/wwdc2005/

Overview

- Introduction
  - Why are you here?
- Part I—Design
  - What is type identification?
  - How are types represented?
  - How are types organized?
- Part II—Practice
  - How do you use this stuff?

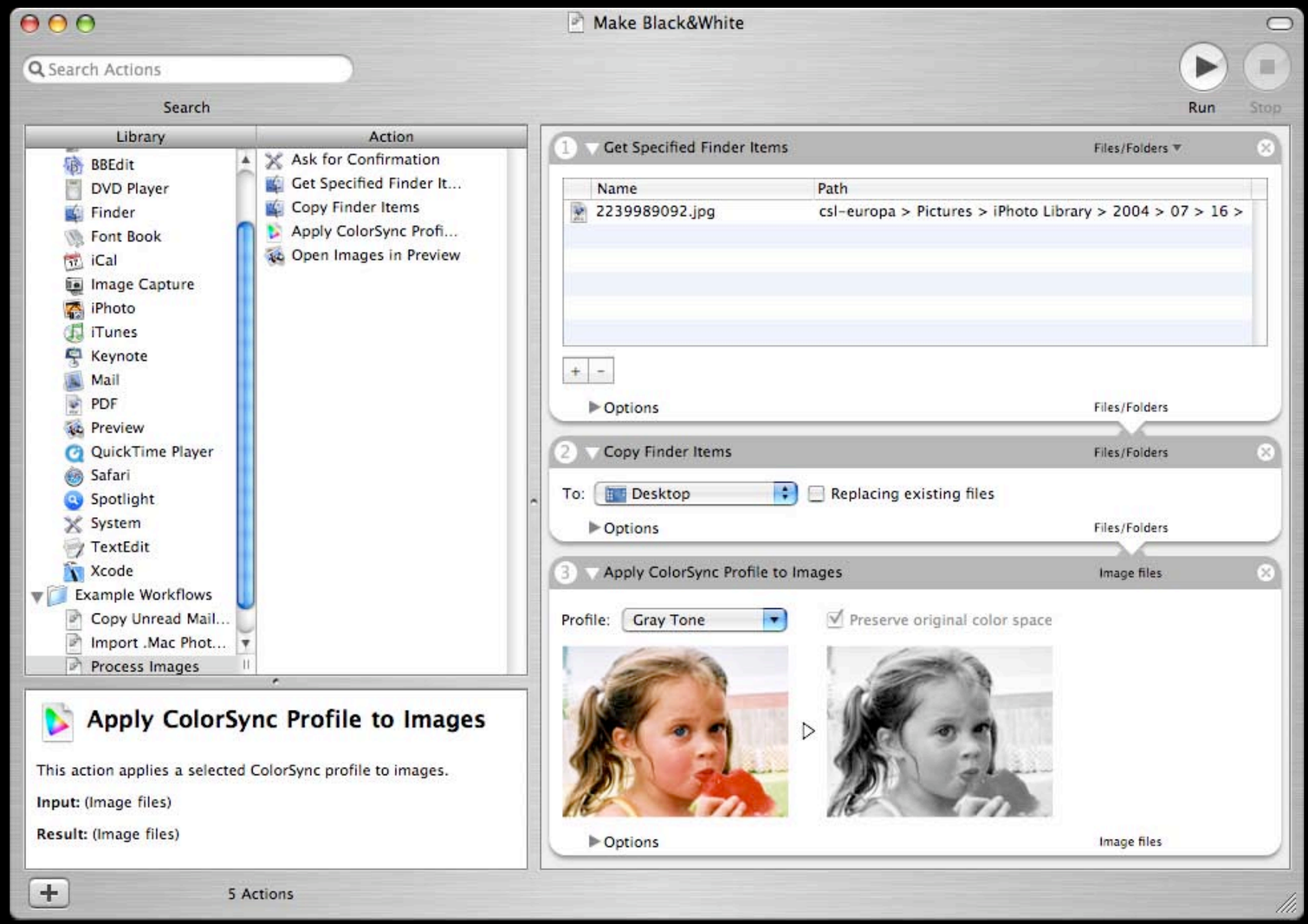
Why Are You Here?

Spotlight



Why Are You Here?

Automator



Why Are You Here?

- Core Image
  - Image Data I/O
- Carbon API
  - Pasteboard and Drag
  - Translation Services
  - Navigation Services
- Launch Services
  - Simplified Application Document Types
  - Binding Preferences
  - Drag and Drop Service Invocation

Technology Map

Finder

Mail

Automator

Your App

Spotlight

Carbon

Core Image

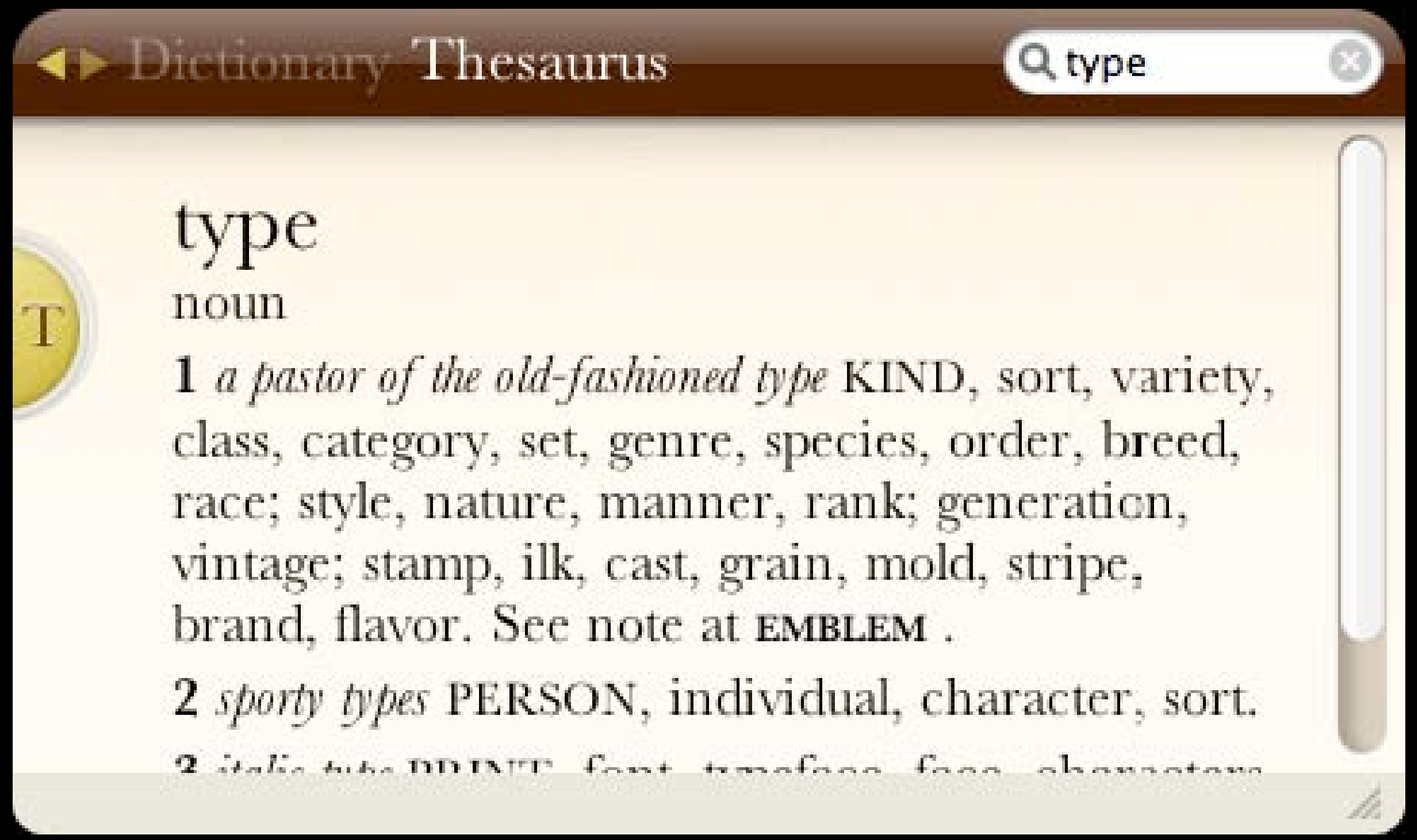
Launch Services

Uniform Type Identifiers

Part I—Design

http://developer.apple.com/wwdc2005/

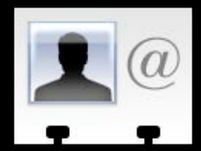
What Is Type Identification?



What Kinds of Things?

Pasteboard

Files



People

Your Application

Folders

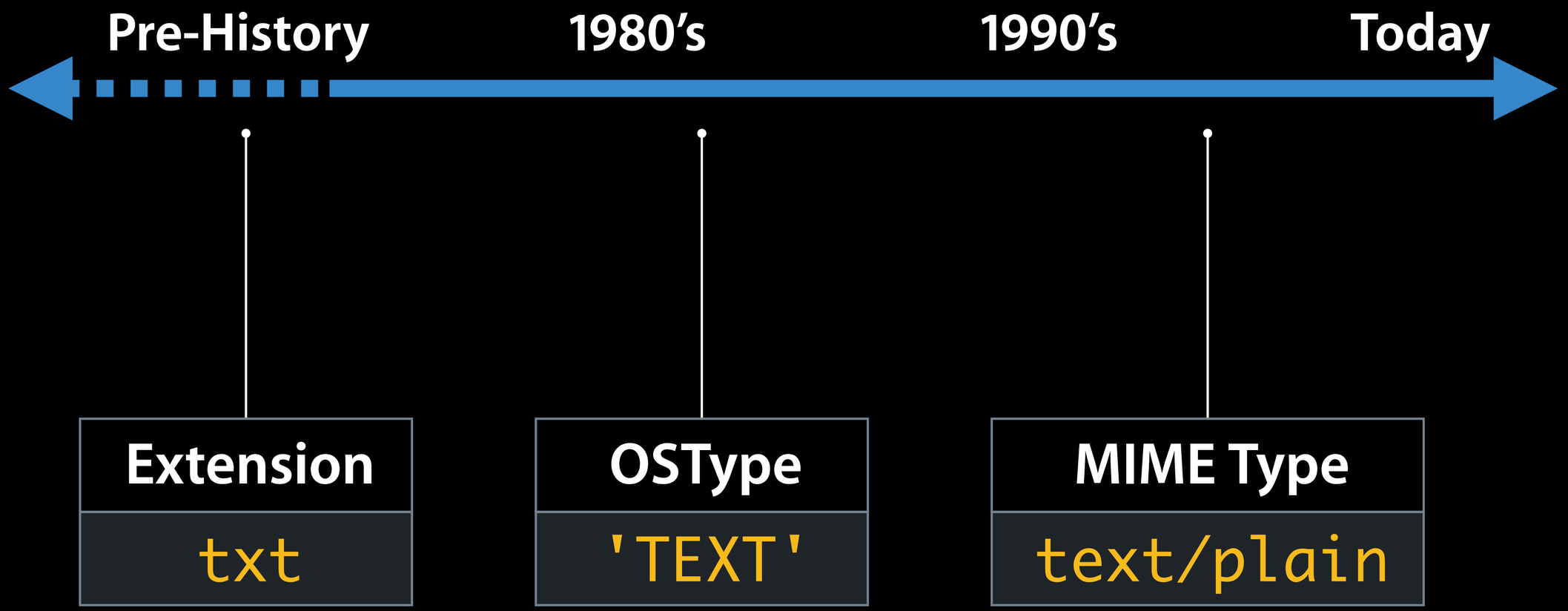
Email

Volumes

Events

Internet

Type Identifiers Through the Ages



Uniform Type Identifiers

- Reverse DNS naming convention prevents collisions
- Freely extensible by Apple and developers—no bureaucracy
- Verbose names are easy to understand
- The `public` domain is reserved by Apple for standard types

`public.text`

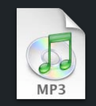
`com.apple.application`

`public.c-plus-plus-source`

How Are Types Represented?

How Are Types Represented?

`public.mp3`

Description:	MP3 audio
Icon:	
Extension:	`mp3`
MIME Types:	`audio/mpeg` `audio/mp3`
OSTypes:	`'MPG3'` `'MP3 '`

Type Conformance

public.content

public.audiovisual-content

public.audio

public.mp3

Description:	MP3 audio

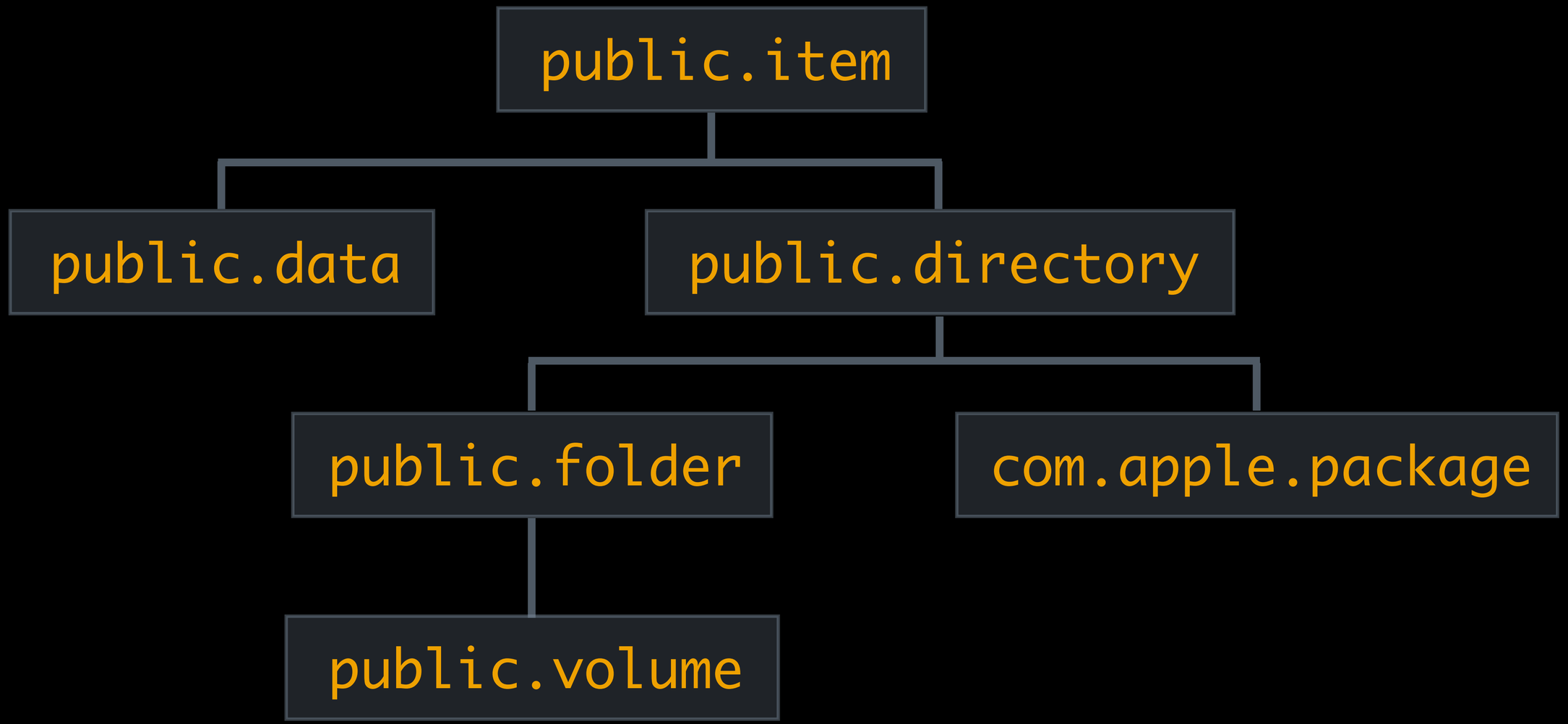
How Are Types Organized?

How Are Types Organized?

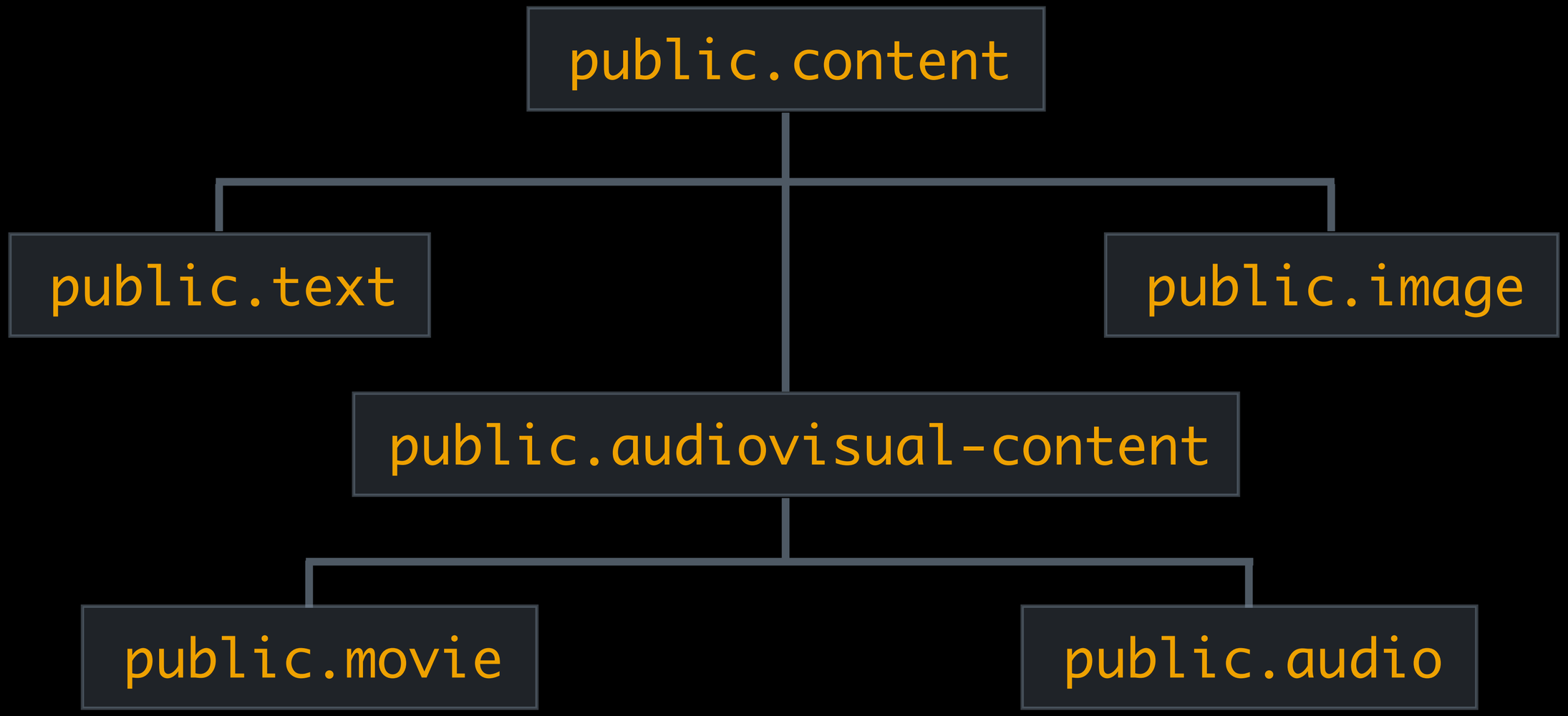
Multiple Hierarchies

- Physical hierarchy
  - Low-level, technical view
- Functional hierarchy
  - High-level, user view
- Every type has a place in both hierarchies

The Physical Hierarchy



The Functional Hierarchy



The Functional Hierarchy

public.content

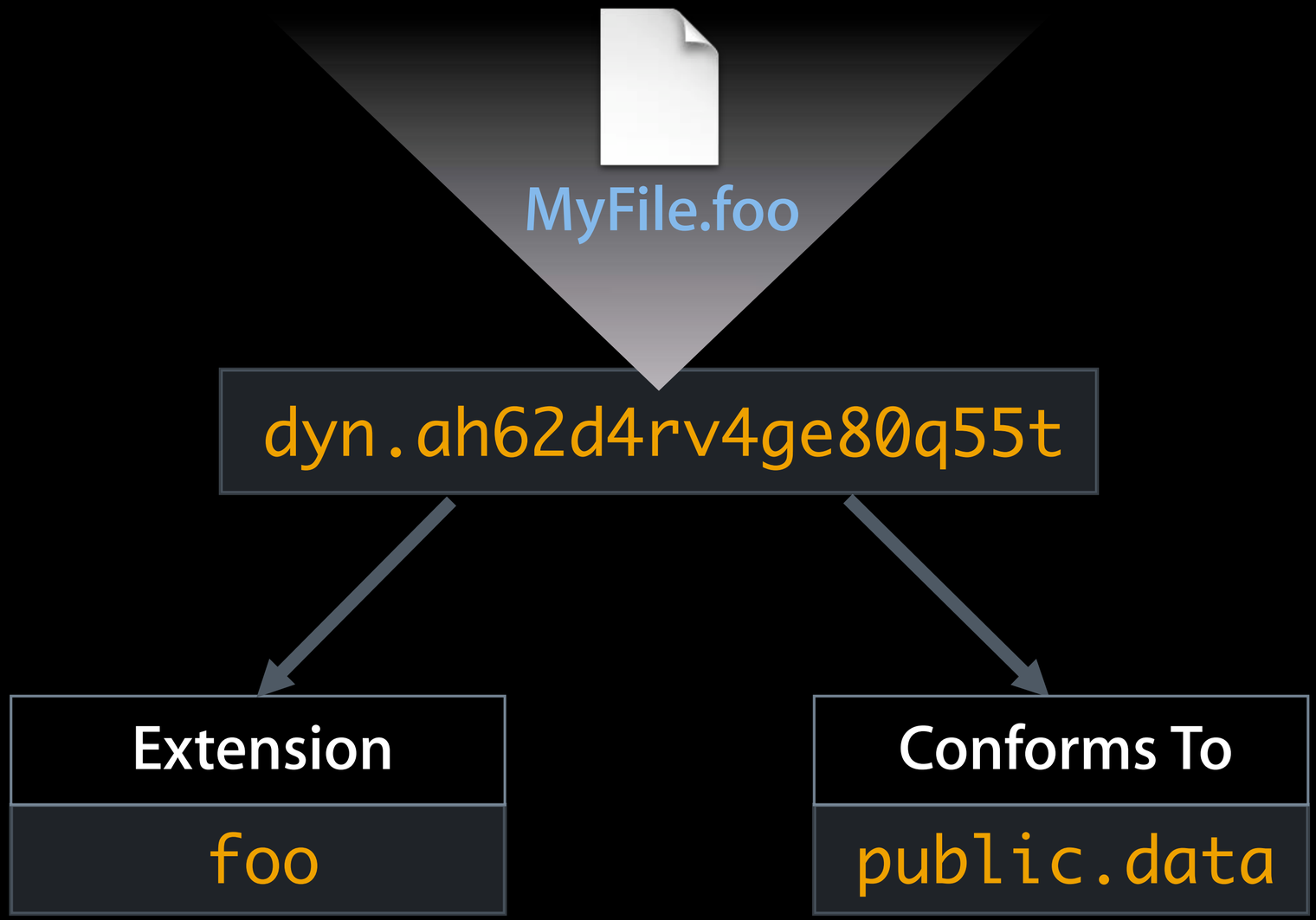
public.message

public.calendar-event

public.contact

public.archive

Dynamic Type Identifiers



Part II—Practice

http://developer.apple.com/wwdc2005/

Two Key Usage Patterns

- Work with existing types to
  - Adopt new carbon APIs (pasteboard, translation, navigation)
  - Simplify file system interaction
  - Display Spotlight search results
- Declare new types for better Tiger integration
  - Writing a Spotlight importer
  - Creating an Automator action plugin

Working With Existing Types

- Over 150 type declarations are built-in to Mac OS X Tiger
- Abstract types structure the hierarchy
- Concrete types for many standard or common data and document formats
- See string constants in header `UTCoreTypes.h`

```
/*  kUTTypeJPEG
 *
 *    JPEG image
 *
 *    UTI: public.jpeg
 *    conforms to: public.image
 */
extern const CFStringRef kUTTypeJPEG;
```

Core API

- Conformance test
- Convert between UTI and “type tags”—MIME, extenison, etc.
- Fetch type attributes
- See header UTType.h

```
Boolean conforms;
CFStringRef extension, description;

conforms = UTTypeConformsTo( myFileUTI, kUTTypeImage );

extension = UTTypeCopyPreferredTagWithClass( kUTTypeJPEG,
              kUTTagClassExtension );

description = UTTypeCopyDescription( kUTTypeHTML );
```

Launch Services Named Attributes

- Fetch one or many attributes for any file system object
- Key attributes
  - Content Type (UTI)
  - Display Name
  - Kind String

```
CFStringRef contentTypeUTI;

OSStatus err = LSCopyItemAttribute(
                    &fileFSRef,
                    kLSRolesAll,
                    kLSItemContentType,
                    &contentTypeUTI );
```

UTIs and Document Type Claims

- Simplify your Info.plist
- Take advantage of type conformance
- No more wild-card claims

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>LSItemContentTypes</key>
        <array>
            <string>public.data</string>
        </array>
        <key>CFBundleTypeRole</key>
        <string>Viewer</string>
    </dict>
</array>
```

Demo

Leveraging UTIs in SimpleText

http://developer.apple.com/wwdc2005/

Declaring New Types

- Types are declared in your bundle's Info.plist file
  - Applications
  - Spotlight importers
  - Automator action bundles
- Types you own are "exported"
- Types you depend on are "imported"
  - Exception: system-declared types
- Description and icon are loaded from bundle resources

Example Type Declaration

```
<key>UTExportedTypeDeclarations</key>
<array>
    <dict>
        <key>UTTypeIdentifier</key>
        <string>com.mycompany.myimageformat</string>
        <key>UTTypeConformsTo</key>
        <string>public.image</string>
        <key>UTTypeDescription</key>
        <string>My Image</string>
        <key>UTTypeTagSpecification</key>
        <dict>
            <key>public.filename-extension</key>
            <string>myimage</string>
            </array>
            <key>public.mime-type</key>
            <string>image/x-myimageformat</string>
        </dict>
    </dict>
```

Registering With Launch Services

- Register your bundle with Launch Services to activate
  - Bundle Identifier and icon
  - Document bindings
  - Type declarations
- Register applications by browsing, copying, or launching
- Register importers and actions by dragging install
  - Library/Spotlight
  - Library/Automator
- Installers can also call any variant of `LSRegister()`
- Touch mod date when editing Info.plist in place!

The Launch Services Database

- Database file paths: /Library/Caches/com.apple.LaunchServices*
- Each user has a separate database (new for Tiger)
- Debug your Info.plist using the lsregister tool

```
% find /System/Library/Frameworks -name lsregister

% lsregister -dump

% lsregister -kill -seed

% lsregister -f <path>
```

New Launch Services API

- Named attributes for file system items
- New `LSOpen()` functions
  - Replacement for `LaunchApplication()`
  - Returns process serial number(s)
- New user binding preferences functions to “get” and “set”
  - Document type handlers (UTI-based)
  - URL scheme handlers
  - Replacement for Internet Config
- Helpful headers: LSOpen.h, LSInfo.h

Drag and Drop Service Invocation

- Drag almost anything to application icons in the dock
  - Documents (of course)
  - URL text
  - Data selection handled by a send only service in target application
- Full visual feedback provided by the dock
- Upon drop, service is invoked in the target application

Demo

Drag and Drop Service Invocation

http://developer.apple.com/wwdc2005/

Behind the Scenes Details…

- A new Apple Event
  - The `kAEOpenContents` event—`'ocon'`
  - Direct object is a list of data items to open
- How to handle the data
  - Open it in a new untitled window
  - Perform some other command using the data
- Steps for adoption
  - Advertise content services in your Info.plist (NSServices)
  - Default `'ocon'` system handler will invoke your service
  - Override the default handler if you wish

Summary—Your Action Items

- Use uniform type identifiers
  - Simplify your code
  - Improve user experience
- Declare new UTIs to integrate with key Tiger features
- Consider new Launch Services APIs
  - Access file system item named attributes
  - Control document and URL binding preferences
  - Open documents, URLs, and applications with new options
- Adopt enhanced drag and drop support in the dock

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Q&A

Mark Malone

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference